**Build Tool:**

A build tool is a software utility that automates the process of transforming source code into a deployable, executable program. It simplifies and automates tasks like compiling, testing, packaging, and deploying software, freeing up developers to focus on writing code rather than repetitive tasks.

What Build Tools Do:

* **Compilation:**Build tools compile source code (like Java, Python, or C++) into machine-readable code.
* **Dependency Management:**They handle the process of finding, installing, and managing dependencies (libraries and frameworks) required by the software.
* **Packaging:**They package the compiled code and dependencies into a deployable format, like a JAR file (Java), a .deb file (Linux), or an executable file (Windows).
* **Testing:**They can automate running unit tests and integration tests to ensure the software is working correctly.
* **Deployment:**They can automate the process of deploying the packaged software to a server or other target environment.

Why Use Build Tools?

* **Increased Productivity:**By automating tasks, build tools save developers time and effort, allowing them to focus on more complex and creative aspects of the development process.
* **Consistency:**They ensure that the build process is consistent across different environments and developers, reducing errors and inconsistencies.
* **Reduced Errors:**By automating tasks, build tools minimize the risk of human error, particularly when dealing with complex dependencies or large projects.
* **Improved Collaboration:**They facilitate collaboration by providing a standardized and automated build process for all team members.
* **Integration with CI/CD Pipelines:**They can be integrated into Continuous Integration/Continuous Delivery (CI/CD) pipelines to automate the entire software delivery process, from code changes to deployment.
* **Build Automation:**Maven automates the process of compiling, testing, and packaging Java projects, making the development workflow more efficient.
* **Dependency Management:**Maven simplifies the process of managing project dependencies, including third-party libraries and JAR files.
* **Project Object Model (POM):**Maven uses a POM (typically a pom.xml file) to define project metadata, build configurations, and dependencies.
* **Plugins:**Maven employs plugins to extend its functionality and automate various tasks, like generating documentation or running tests.
* **Central Repository:**Maven uses a central repository to store and manage dependencies for projects.
* **Standardization:**Maven provides a standard way to build and manage projects, promoting consistency across teams and projects.
* **Beyond Java:**While primarily used for Java, Maven can also be used for projects in other languages like C#, Ruby, and Scala.
* **Release Management:**Maven can be used for version control, documentation, and reporting, making it a valuable tool for project management.
* **1. POM Configuration:**  
  The core of Maven is the Project Object Model (POM), which defines the project's metadata, dependencies, and build configurations.
* **2. Build Lifecycle:**  
  Maven follows a build lifecycle with phases (e.g., clean, compile, test, package) that automate the building process.
* **3. Plugins:**  
  Maven plugins extend its functionality and automate tasks like compiling, testing, packaging, and generating documentation.
* **4. Dependency Management:**Maven automatically downloads and manages dependencies specified in the POM.
* **5. Build Artifacts:**  
  Maven generates build artifacts (e.g., JAR files) that can be deployed or distributed.

[Maven](https://maven.apache.org/) is an open-source build automation and project management tool widely used for Java applications. As a build automation tool, it automates the source code compilation and dependency management, assembles binary codes into packages, and executes test scripts. Maven translates and packages your source code so that it becomes an executable application. Maven can manage a project's build, reporting and documentation from a central piece of information.

Using Maven, you can create Java deliverables like **JAR, EAR, and WAR** files. The pom.xml file helps you to do these tasks. These XML files contain your Java project’s name, group ID, and other valuable information. Based on this information, Maven creates the JAR (Java archives) files and transfers them to the Maven repository.

**1. Project Object Model (POM)**

A POM is the basement of the Maven framework. It’s a type of XML file that accommodates data from your project and the configuration details. It includes the project, group ID, POM model version, artifact ID (project ID), and version. The project is the key element of your XML file. Group ID means the ID of the group to which your project belongs. Here, the version informs you about the number of your project releases.

**2. Maven repositories and dependencies**

The Maven repository stores all your projects’ jars, plugins, library jars, other artifacts, and the dependencies are the third-party software required by your project. You will find three types of repositories in Maven – **local, central, and remote**. The local repository stores all the Maven dependencies.

The Maven community handles the **central** repository. If you don’t find a dependency in your local repo, you can find it in the central repo. Again the developers can utilize their customized repository. That is known as the remote repository.

**Build lifecycle is made up of different phases. The default lifecycle consists of the following phases:**

* **Validate**: It authorizes your project’s correctness and ensures necessary data are available.
* **Compile**: It compiles the source code of your project.
* **Test**: It tests your compiled source codes with the unit testing frameworks.
* **Package**: In this phase, your source code will be packaged as a deliverable.
* **Verify**: It ensures your code’s quality with an integration test.
* **Install**: Your code will be installed in the local repository.
* **Deploy**: Finally, the code is ready to share with the other developers.

An archetype is a model, pattern, or symbol that is universally understood and used as a prototype for others to copy, emulate, or pattern.

Maven Archetype is a Maven plugin that makes it possible to create a project structure based on a template. These archetypes are essentially project templates that Maven generates when you create a new project. Archetype is a Maven project templating toolkit

| **Feature** | **Compile** | **Build** | **Run (Execute)** |
| --- | --- | --- | --- |
| **Purpose** | Translate source code to machine/bytecode | Assemble all components into a deployable unit | Execute the ready program |
| **Input** | Source code files (.c, .java, etc.) | Source code, dependencies, resources | Executable files (.exe, .jar, .pyc, etc.) |
| **Output** | Object files, bytecode | Executable program, library, or package | Program's desired behavior/output |
| **Tool** | Compiler (e.g., GCC, Javac) | Build tool (e.g., Make, Maven, Gradle) | Operating System's loader, interpreter, JVM |
| **Scope** | Single file/unit translation | Entire project assembly | Single program execution |
| **Dependency** | Needs source code | Needs source code, compilers, linkers, assets | Needs a built/compiled program |